

## Von der Programmiersprache zu den Nullen und Einsen

Dass der Code unserer Dosenampel durch einen **Binärkode** (also einen Code mit zwei verschiedenen Zeichen; in unserem Falle die Zeichen 0 und 1) dargestellt werden kann, haben wir in den letzten Stunden gesehen. Heute sehen wir nun endlich, dass auch ganz „normale“ Programmiersprachen in logischen Schritten in Nullen und Einsen umgewandelt werden können. Wir verfolgen dabei in einfachen Schritten den **Weg von der Programmiersprache hin zu den Nullen und Einsen**, mit denen Computer oder Smartphones etc. heutzutage im Kern – immer noch – arbeiten.

### Höhere Programmiersprachen

```
int main() {
    int a = 2;
    int b = 3;
    int c = a + b;
    return c;
}
```

Wenn man heute von „ProgrammiererInnen“ spricht, meint man Leuten, die **Software** erstellen (Betriebssysteme, Spiele, PC-Programme, ...). Dazu benutzen sie **höhere Programmiersprachen**. Es ist nicht unüblich, dass ProgrammierInnen mehrere Sprachen beherrschen (z. B. C++, Java, ...). **Programmiersprachen** sind für uns Menschen **gut lesbar**, da sie meist **englische Wörter** beinhalten, die dann den **Programmcode** ergeben. Die Programme, in denen man programmiert, nennt man auch „**integrierte Entwicklungsumgebung**“. Sie **können den Programmcode unter anderem direkt in funktionierende Software umwandeln**. Von diesem Prozess merkt man nichts.



... kann aber auch - rein aus Neugier - umgewandelt oder direkt geschrieben werden in Assemblersprachen ...

### Von Assemblersprachen zu Binärkode

#### **zugehöriger Assemblercode**

```
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-4], 2
mov DWORD PTR [rbp-8], 3
mov eax, DWORD PTR [rbp-8]
mov edx, DWORD PTR [rbp-4]
add eax, edx
mov DWORD PTR [rbp-12], eax
mov eax, DWORD PTR [rbp-12]
pop rbp
ret
```

Eine **Assemblersprache** ist für uns Menschen zwar auch noch **gut lesbar**, ist aber schon sehr an den verbauten **Prozessor im PC gebunden**. Sie ist deshalb eine **maschinenorientierte Sprache**. Im Prozessor werden Nullen und Einsen „verrechnet“. Da es jedoch zig verschiedene Prozessoren (Intel, AMD, ...) und noch mehr Modelle gibt, **sieht die Assemblersprache immer anders aus**. Die Programme, in denen man in Assemblersprache schreibt, heißen auch **Assembler**. Sie **können den Assemblercode direkt in funktionierende Software umwandeln**. Von diesem Prozess merkt man nichts. Assemblersprachen sind älter als die höheren Programmiersprachen, mit denen man heutzutage programmiert. Aber sie helfen uns dabei, den Weg zu den Nullen und Einsen besser zu verstehen!

#### **Zum Verständnis schauen wir uns das Beispiel links also genauer an!**

Assemblersprachen benutzen in ihrem Code **mnemonische Symbole** wie z. B. „mov“, die dem Prozessor letztlich sagen, was mit den Nullen und Einsen passieren soll. Unser Befehl wird wohl so etwas tun wie die Nullen und Einsen im Prozessor an einen anderen Ort zu kopieren (englisch „to move“). Diese mnemonischen Symbole können wir uns **gut einprägen** (das Wort „mnemo“ kann auch mit „Merkhilfe“ übersetzt werden).

Intel x86 Assembler Instruction Set Opcode Table

ADD Eb Gb 90	ADD Gb Eb 91	ADD Gb Eb 92	ADD Gb Eb 93	ADD AL Ib 94	ADD eAX Iv 95	PUSH ES 96	POP ES 97	OR Eb Gb 98	OR Gb Eb 99	OR Gb Eb 9A	OR Gb Eb 9B	OR AL Ib 9C	OR eAX Iv 9D	PUSH CS 9E	TWOBYTE 9F
ADC Eb Gb 10	ADC Gb Eb 11	ADC Gb Eb 12	ADC Gb Eb 13	ADC AL Ib 14	ADC eAX Iv 15	PUSH SS 16	POP SS 17	SBB Eb Gb 18	SBB Gb Eb 19	SBB Gb Eb 1A	SBB Gb Eb 1B	SBB AL Ib 1C	SBB eAX Iv 1D	PUSH DS 1E	POP DS 1F
AND Eb Gb 20	AND Gb Eb 21	AND Gb Eb 22	AND Gb Eb 23	AND AL Ib 24	AND eAX Iv 25	ES: 26	DAA 27	SUB Eb Gb 28	SUB Gb Eb 29	SUB Gb Eb 2A	SUB Gb Eb 2B	SUB AL Ib 2C	SUB eAX Iv 2D	CS: 2E	IAS 2F
XOR Eb Gb 30	XOR Gb Eb 31	XOR Gb Eb 32	XOR Gb Eb 33	XOR AL Ib 34	XOR eAX Iv 35	SS: 36	AAA 37	CMP Eb Gb 38	CMP Gb Eb 39	CMP Gb Eb 3A	CMP Gb Eb 3B	CMP AL Ib 3C	CMP eAX Iv 3D	DS: 3E	AAS 3F
INC eAX 40	INC eCX 41	INC eDX 42	INC eBX 43	INC eSP 44	INC eSI 45	INC eDI 46	DEC eAX 47	DEC eCX 48	DEC eDX 49	DEC eBX 4A	DEC eSP 4B	DEC eSI 4C	DEC eDI 4D	DEC eSI 4E	DEC eDI 4F
PUSH eAX 50	PUSH eCX 51	PUSH eDX 52	PUSH eBX 53	PUSH eSP 54	PUSH eSI 55	PUSH eDI 56	POP eAX 57	POP eCX 58	POP eDX 59	POP eBX 5A	POP eSP 5B	POP eSI 5C	POP eDI 5D	POP eSI 5E	POP eDI 5F
PUSHA 60	POPA 61	BOUND Gv Ma 62	ARPL Ew Gv 63	FS: 64	GS: 65	OPSIZE: 66	PUSH Iv 67	IMUL Ev Gv Iv 68	PUSH Ib 69	IMUL Gv Ev Ib 6A	INSD Yz DX 6B	INSW Yz DX 6C	OUTSB DX Xb 6E	OUTSW DX Xv 6F	
JO Jb 70	JNO Jb 71	JB Jb 72	JNB Jb 73	JZ Jb 74	JNZ Jb 75	JBE Jb 76	JA Jb 77	JS Jb 78	JNS Jb 79	JP Jb 7A	JMP Jb 7B	JL Jb 7C	JLE Jb 7D	JLE Jb 7E	JNLE Jb 7F
ADD Eb Iv 80	ADD Iv Eb 81	SUB Eb Iv 82	SUB Iv Eb 83	TEST Eb Gb 84	TEST Eb Gb 85	XCHG Eb Gb 86	XCHG Ev Gv 87	MOV Eb Gb 88	MOV Ev Gv 89	MOV Gb Eb 8A	MOV Gv Eb 8B	MOV Ev Sw 8C	MOV Sw Ev 8D	LEA Sw M 8E	MOV Sw Ev 8F
NOP 90	XCHG eAX 91	XCHG eAX 92	XCHG eAX 93	XCHG eAX 94	XCHG eAX 95	XCHG eAX 96	XCHG eAX 97	CBW 98	CWD 99	CALL 9A	WAIT 9B	PUSHF 9C	POPF 9D	SAHF 9E	LAHF 9F
MOV AL Gv A1	MOV eAX Gv A2	MOV eAX AL A3	MOV eAX A1 A4	MOVSB Xb Yb A5	MOVSW Xv Yv A6	CMPSB Xb Yb A7	CMPSW Xv Yv A8	TEST AL Ib A9	TEST eAX Iv AA	STOSB Yb AL AB	STOSW Yv eAX AC	LODSB AL Xb AD	LODSW eAX Xv AE	SCASB AL Yb AF	SCASW eAX Yv B0
MOV AL Ib B0	MOV CL Ib B1	MOV DL Ib B2	MOV BL Ib B3	MOV AH Ib B4	MOV CH Ib B5	MOV DH Ib B6	MOV BH Ib B7	MOV eAX Iv B8	MOV eCX Iv B9	MOV eDX Iv BA	MOV eBX Iv BB	MOV eSP Iv BC	MOV eBP Iv BD	MOV eSI Iv BE	MOV eDI Iv BF
#2 Eb Ib C0	#2 Ev Ib C1	RETN C2	RETN C3	LES Gv Mp C4	LDS Gv Mp C5	MOV Eb Ib C6	MOV Ev Ib C7	ENTER Iv Ib C8	LEAVE Iv Ib C9	RETF Iv Ib CA	RETF Iv Ib CB	INT3 Ib CC	INT1 Ib CD	INTO Ib CE	IRET Ib CF
#2 Eb 1 D0	#2 Ev 1 D1	#2 Eb CL D2	#2 Ev CL D3	AAM Ib D4	AAD Ib D5	SALC D6	XLAT D7	ESC 0 D8	ESC 1 D9	ESC 2 DA	ESC 3 DB	ESC 4 DC	ESC 5 DD	ESC 6 DE	ESC 7 DF
LOOPNZ Jb E0	LOOPZ Jb E1	LOOP Jb E2	JCZJ Jb E3	IN AL Ib E4	IN eAX Ib E5	OUT Ib AL E6	OUT Ib eAX E7	CALL Jz E8	JMP Jz E9	JMP Ap EA	JMP Jb EB	IN AL DX EC	IN eAX DX ED	OUT DX AL EE	OUT DX eAX EF
LOCK: F0	INT1 F1	REPNE: F2	REP: F3	HLT F4	CMC F5	#3 Eb F6	#3 Ev F7	CLC F8	STC F9	CLI FA	STI FB	CLD FC	STD FD	#4 INCDEC FE	#5 INCDEC FF

Alle mnemonischen Symbole sind wiederum durch Zahlen - so genannte Opcodes - durchnummeriert. **Unser Befehl „mov“ hat bei diesem Prozessor die Zahl bzw. den Opcode „B8“**, weil es der Hersteller des Prozessors so festgelegt hat. Auf dem Weg zur funktionierenden Software wandelt der Assembler nun alle mnemonischen Symbole im Programmcode automatisch in Opcodes - also Zahlen - um. Neben den mnemonischen Symbolen werden auch die anderen Bestandteile des Assemblercodes umgewandelt.

Doch warum ist „B8“ eine Zahl? Ganz einfach! Diese **hexadezimale Zahl** kann ganz einfach auch in Nullen und Einsen umgewandelt bzw. dargestellt werden. **Die hexadezimalen Opcodes sind also ein wichtiger „Schlüssel“ zu den Nullen und Einsen.**

Dez	Hex	Okt	Binär
176	B0	260	10110000
177	B1	261	10110001
178	B2	262	10110010
179	B3	263	10110011
180	B4	264	10110100
181	B5	265	10110101
182	B6	266	10110110
183	B7	267	10110111
<b>184</b>	<b>B8</b>	<b>270</b>	<b>10111000</b>
185	B9	271	10111001
186	BA	272	10111010
187	BB	273	10111011
188	BC	274	10111100
189	BD	275	10111101
190	BE	276	10111110
191	BF	277	10111111

Anhand einer Tabelle wandeln wir nun unseren hexadezimalen Opcode ganz einfach in Nullen und Einsen um. „B8“ ist also „10111000“.

**Schön und gut. Wir können unseren Programmcode in Nullen und Einsen umwandeln. Was passiert nun?**

Einzelne Opcodes (oder Opcodes mit zusätzlichen Angaben wie z. B. Adressen) in binärer Darstellung stellen direkt **Maschinenbefehle** dar. Maschinenbefehle sagen dem Prozessor, wie er Nullen und Einsen verarbeiten soll, so dass wir - über weitere Zwischenschritte - Informationen auf dem Bildschirm sehen oder uns z. B. Musik am Computer anhören können. **Mnemonische Symbole in ihrer menschenlesbaren Form geben uns zumindest eine Ahnung, welche Aktion der spätere Maschinenbefehl dann konkret auslöst (siehe oben, „mov“). Der Kreis schließt sich also.**

**Zusammenfassung**

Hier nochmals der gesamte Weg vom Programmcode einer höheren Programmiersprachen hin zu den Nullen und Einsen (am Beispiel der Programmiersprache C):

```
int main() {
    int a = 2;
    int b = 3;
    int c = a + b;
    return c;
}
```



zugehöriger C-Code	zugehöriger Assemblercode	Maschinencode (hexadezimal)	oder Maschinencode (binär)
<code>int main() {</code>	<code>push rbp</code>	55	01010101
<code>int a = 2;</code>	<code>mov rbp, rsp</code>	48 89 E5	01001000 10001001 11100101
<code>int b = 3;</code>	<code>mov DWORD PTR [rbp-4], 2</code>	C7 45 FC 02	
<code>int c = a + b;</code>	<code>mov DWORD PTR [rbp-8], 3</code>	C7 45 F8 03	
	<code>mov eax, DWORD PTR [rbp-8]</code>	8B 45 F8	
	<code>mov edx, DWORD PTR [rbp-4]</code>	8B 55 FC	
	<code>add eax, edx</code>	01 D0	
	<code>mov DWORD PTR [rbp-12], eax</code>	89 45 F4	
<code>return c;</code>	<code>mov eax, DWORD PTR [rbp-12]</code>	8B 45 F4	
<code>}</code>	<code>pop rbp</code>	5D	
	<code>ret</code>	C3	

**In der Tabelle auf dem nächsten Arbeitsblatt könnt ihr selbst den hexadezimalen Maschinencode in binären Maschinencode mit Nullen und Einsen umwandeln! Ich bin gespannt, was euch bei diesem Arbeitsauftrag auffällt!**

Quellen:

- <https://de.wikipedia.org/wiki/Compiler> (17.06.2020)
- <https://de.wikipedia.org/wiki/Bin%C3%A4rcode> (17.06.2020)
- [http://fbmathe.bbs-bingen.de/informatik/C\\_plusplus/grundbegriffe.htm](http://fbmathe.bbs-bingen.de/informatik/C_plusplus/grundbegriffe.htm) (17.06.2020)
- <https://de.wikipedia.org/wiki/Opcode> (17.06.2020)
- <http://sparksandflames.com/files/x86InstructionChart.html> (17.06.2020)
- <https://ascii-tabelle.org/> (17.06.2020)
- <https://de.wikipedia.org/wiki/Maschinensprache> (17.06.2020)
- <http://www.virtualuniversity.ch/software/ascii/> (17.06.2020)
- [http://mikrocontroller.rahm-home.de/docfiles/Uebungen/2\\_3\\_1\\_Maschinensprache\\_und\\_Assemblercode.pdf](http://mikrocontroller.rahm-home.de/docfiles/Uebungen/2_3_1_Maschinensprache_und_Assemblercode.pdf) (17.06.2020)
- [https://www.kstbb.de/informatik/oo/06/6\\_1\\_Programmiersprachen\\_der\\_ersten\\_bis\\_dritten\\_Generation.html](https://www.kstbb.de/informatik/oo/06/6_1_Programmiersprachen_der_ersten_bis_dritten_Generation.html) (17.06.2020)
- <https://www.it-talents.de/blog/it-talents/der-assembler-befehlssatz> (17.06.2020)
- <https://www.publicdomainpictures.net/de/view-image.php?image=137430&picture=cpu> (17.06.2020)